A Distributed Component Framework for Science Data Product Interoperability

Daniel Crichton, Steven Hughes, Sean Kelly, Sean Hardman
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109

Abstract - Correlation of science results from multi-disciplinary communities is a difficult task. Traditionally, data from science missions is archived in proprietary data systems that are not interoperable. The Object Oriented Data Technology (OODT) task at the Jet Propulsion Laboratory is working on building a distributed product server as part of a distributed component framework to allow heterogeneous data systems to communicate and share scientific results. These components communicate using a standard metadata interchange language. This provides an excellent vehicle for turning data into information and allowing for data in unique formats to be correlated and exchanged. Advances in Internet and distributed object technologies provide an excellent framework for sharing data across multiple data systems. The product server component of the OODT framework allows for results to be interchanged between native data system formats and the framework using an XML-based query language. The product server component wraps data system interfaces, which abstracts away the data system unique interfaces, and provides a scalable architecture by providing query handlers that facilitate the interchange of queries and results. This paper, the second in a series on the OODT task, focuses on the development of the product server component using the Planetary Data System (PDS) as an example system. This continues the discussion of an enterprise framework that allows for data system interoperability across multiple science disciplines.

I. Introduction

Science data has continued to devolve into a large set of highly fragmented distributed data systems. These systems are heterogeneous and geographically distributed making interoperability and integration difficult. Furthermore, correlating science data across a multi-disciplinary environment is even more challenging. The Object Oriented Data Technology task at the Jet Propulsion Laboratory is currently researching a distributed framework that will allow for data products from distributed data systems to be located and retrieved.

Data systems across NASA are heterogeneous in nature. They have traditionally followed stove pipe implementations, and there has been very little integration across these systems. The implementations are often unique, and there is no standard mechanisms for data interchange between these systems. However, one NASA system, the Planetary Data System (PDS), has developed an archive standards architecture that provides a good metadata foundation for developing the proposed data interchange mechanism

The Planetary Data System (PDS) manages and archives planetary science data for NASA's Office of Space Science. In existence since the late 1980s, the PDS early on developed a standards architecture that included a formal enterprise model, a means for collecting and associating metadata with science data products, and a peer review process for ensuring data and metadata validity. This active science data archive currently has over five terabytes of data curated by six distributed science nodes and stored and distributed on CD and DVD media. The standards architecture as proven to be critical to maintaining consistency archive across the various science domains represented in the planetary science community and for supporting a level of interoperability across the nodes. For example, the archive includes a diversity of data products from images and time series to spectrum. These products are stored in a variety of data and machine formats and are served from heterogeneous hardware platforms. However the metadata used for searching and describing the resulting data products is consistent due to the standards architecture. The metadata is presented in a common interchange language supported by a common data dictionary.

The PDS experience in developing and enforcing metadata standards as proven to be a critical element in maintaining consistency across the distributed system. The advent of the internet and web technology as now affords an excellent opportunity to exploit this metadata to provide a new level of interoperability both within the PDS nodes as well as with other space science data systems. The OODT task is currently working with the Planetary Data System to provide a data architecture that allows for data products within the PDS to be located and exchanged across the distributed nodes using a common user interface. We also feel that the methodology being developed will directly address the data system interoperability problems now being encountered in general.

II. Architecture

The OODT architecture has several key architectural objectives which include (1) requiring that individual data systems be encapsulated to hide uniqueness; (2) requiring that communication between distributed services use metadata for data interchange; (3) defining a standard data dictionary based on a metadata for describing data resources; (4) providing a solution that is both scalable and extensible; (5) providing a standard mechanism for exchanging data system product results across distributed services.

The product service is part of a larger component framework [16], which includes a query, profile and product server. Profile and product service instances are distributed across the enterprise and manage information and access to a set of data system resources. A key benefit of this architecture is that new service instances can be introduced in order to scale the system. The basic system architecture is illustrated in Figure 1.

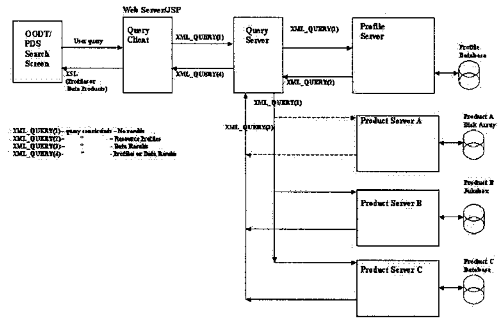


Figure 1: System Architecture

The profile server manages profiles—sets of resource definitions [7]— about distributed data systems and their products. A profile is essentially a metadata description of the resources known at a node in the distributed framework. These resources are either data products archived by an integrated data system, or definitions of other profile nodes that manage metadata about other data systems that can further satisfy the query. Profiles describe nodes of a digraph[1] and may point to other profiles thus representing arcs of the digraph.

Profiles may be grouped and served by more than one profile server. The query components ties this architecture together by providing and managing the traversal of the digraph architecture. The query component also provides the facility to manage concurrent queries across multiple servers order to improve performance.

The product component provides the translation necessary to map a product retrieved from a data-system–dependent environment into a neutral format suitable for exchange between systems. Product components are similar to profile components in that they also represent nodes of the digraph. This allows heterogeneous data systems to be easily added without changing the way their data is stored.

The component architecture described lends itself naturally to a distributed object implementation. We used the Common Object Request Broker Architecture (CORBA) to provide the distributed object framework and to communicate and exchange data in heterogeneous environments using the Internet Inter-ORB Protocol (IIOP) [11]. This activity is currently using an implementation of the CORBA 2.0 standard from Object Oriented Concepts known as Orbacus [12]. Each profile and product server node is defined by a separate object name (or node name). CORBA allows for nodes to be located based on the CORBA naming service that is included in the Orbacus implementation. The naming service allows objects that can satisfy the query to be specified by name so that profiles can identify other profiles or products in their metadata definitions. This enables integration of the described nodes.

Each component of the architecture communicates with other components using the Extensible Markup Language (XML) [14] for the data content running on top of the CORBA implementation. One of the critical requirements of this architecture is to provide interoperability solutions without having to change the implementation of each data system. Our architecture accomplishes this goal by encapsulating each of the individual data systems and then using standard metadata definitions based on XML for interoperability. This allows various implementations ranging from the use of relational and object database management systems to implementations that use flat file and home-grown databases for cataloging and storing data products to exchange information using XML metadata definitions.

We deployed the resource location service entirely in the Java programming language along with CORBA and XML [10]. Java allows for the implementation of the object architecture and allows the framework to be easily extended to integrate new data systems. Java is particularly useful in the design of the product service component which allows new servers to be quickly instantiated by loading additional product translation objects at run time. This will be explained in further detail below.

One of the goals of this architecture is to provide a standard application programmers' interface (API) that will allow for generic science analysis tools to be written that can plug into the architecture to retrieve and correlate data from multiple data sources. This is accomplished using an *n*-tier architecture. Such architectures split the traditional client-server model into three layers: a user interface layer, a domain logic layer, and a storage layer. Abstracting the implementation away from the client allows for the infrastructure to evolve without breaking the tool interfaces. It also moves the domain intelligence to the middleware components which removes the constraint that the tools need to have the knowledge of the protocol and location of data systems in order to query and retrieve data from it. Finally, the *n*-tier architecture also allows the framework to plug in additional services.

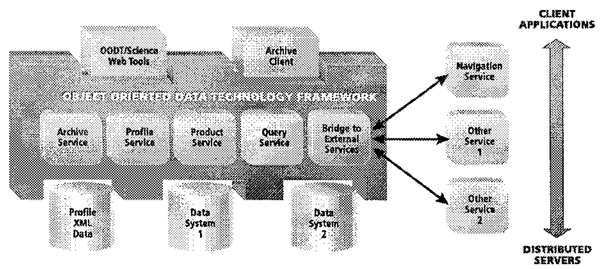


Figure 2: Component framework

In Figure 2 the framework shows general objects that fit into the framework along with bridges to other services that could be potentially added. In this case, a navigation service could be added to allow for images of Jupiter, a constantly moving target, to be found based on metadata that describes the right ascension and declination (RA/DEC) of an image. Also since the navigation service performs coordinate system transformation, this addition illustrates how a software component can use metadata to further increase interoperability between domains.

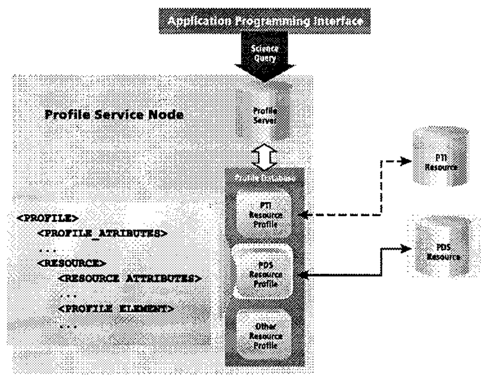


Figure 3: Profile Service Node

Figure 3 illustrates the functioning of one profile service node within the resource location framework. The profile server node is managing resource profiles from multiple disciplines, namely the Palomar Testbed Interferometer (PTI), an astrophysics system, and the Planetary Data System (PDS). A candidate query for an image from the Mars Global Surveyor mission arrives at the node from the API. A candidate PDS resource is then identified by searching the resource profile database. The query system will subsequently use a PDS product delivery service to obtain product information from the resource. Product information may include images, time series data, or simply metadata information. The product delivery service will be described in a subsequent section of the paper.

The component architecture as described focuses on providing a framework for solving complex integration problems across heterogeneous data systems. It addresses the issues of data location, data transformation, and data exchange. The framework provides a scalable architecture that centers around the use of metadata. It also allows for data systems to continue to retain their unique attributes, yet plug into an enterprise architecture that allows for the successful exchange of data content through the use of XML. By using XML this framework is able to impose an inter-disciplinary communication mechanism that allows for data to be shared and exchanged.

III. Query Service

The query component of the framework serves as the starting point for users to retrieve information stored across distributed data nodes. The query component's CORBA interface enables analysis tools to have a programmatic entry point for entering queries and retrieving

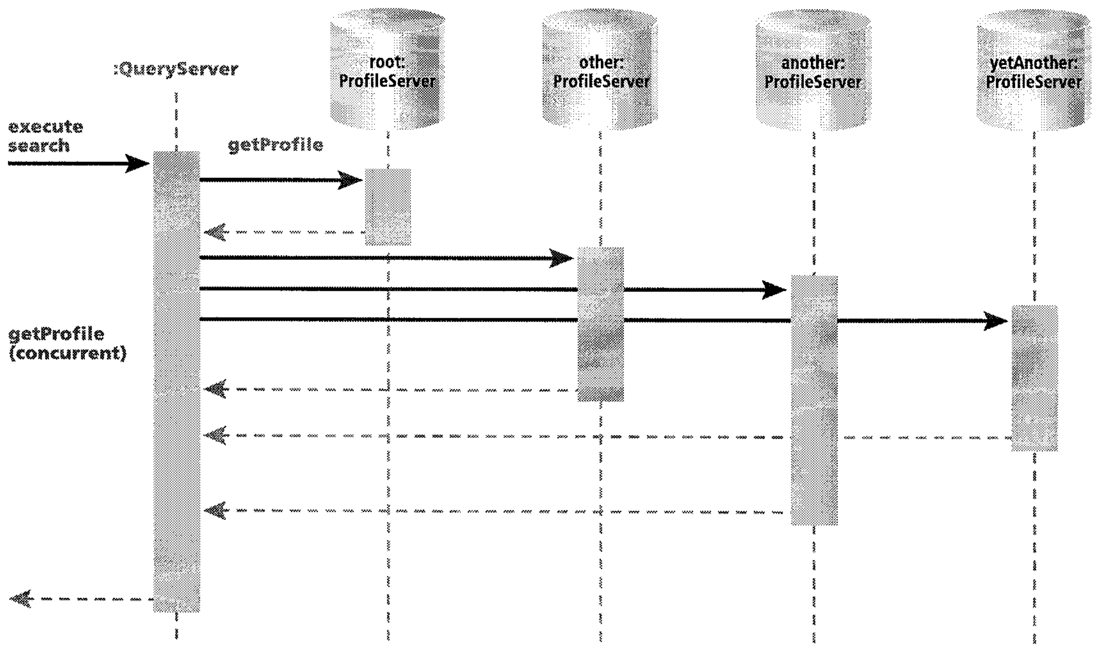


Figure 4: UML Diagram for Profile Search

results. In addition, we have implemented a Java API that wraps the CORBA interface (a C++ API is forthcoming). This enables scientists and engineers to develop their own data analysis tools to access disparate data systems from a single API. As more data systems are added to the framework, existing tools can access the new systems with no changes. Furthermore, multiple user interfaces that access the query component are possible. One such interface that we have developed is a web interface. The web interface uses the Java API to give scientists and engineers immediate access to data systems from any common web browser without any programming or knowledge of what data systems to search.

The query service uses the CORBA naming service to connect to a profile node. In general, searches will enter the directed graph at the root or parent node; however, the query service can enter and search at any point in the graph. Profiles must be registered in order to be searched.

To execute a search, the query component assembles an XML document describing the characteristics of the query. The document includes a header section that describes metadata about the query, such as its title, description, data dictionary, security type, and revision note. These elements indicate to the query service any characteristics, versioning, or special handling required by the query. Also included in the document are preferences on the result, such as how the query should propagate through the digraph, the maximum number of results, the query itself, and a space for results. For example, a query for "TARGET_NAME = MARS" results in the following XML document:

```
<query>
 <queryAttributes>
  <queryId>OODT_Q070321</queryId>
  <queryTitle>PDS DIS Query</queryTitle>
  <queryDesc>PDS DIS Query Example</queryDesc>
  <queryType>QUERY</queryType>
  <queryStatusId>ACTIVE</queryStatusId>
  <querySecurityType>UNKNOWN</querySecurityType>
  <queryRevisionNote>2000-05-12</queryRevisionNote>
  <queryDataDictId>PDS_DS_DD_ V1</queryDataDictId>
 </queryAttributes>
 <queryResultModeId>ATTRIBUTE</queryResultModeId>
<queryPropogationType>BROADCAST
                                        </queryPropogationType>
 <queryPropogationLevels>N/A</queryPropogationLevels>
 <queryMaxResults>100</queryMaxResults>
 <queryResults>0</queryResults>
<queryKWQString>TARGET_NAME=MARS
                                        </queryKWQString>
 <querySelectSet></querySelectSet>
 <queryFromSet></queryFromSet>
 <queryWhereSet>
  <queryElement>
   <tokenRole>elemName</tokenRole>
   <tokenValue>TARGET_NAME</tokenValue>
  </queryElement>
  <queryElement>
   <tokenRole>LITERAL</tokenRole>
   <tokenValue>MARS</tokenValue>
  </queryElement>
  <queryElement>
   <tokenRole>RELOP</tokenRole>
   <tokenValue>EQ</tokenValue>
  </queryElement>
 </queryWhereSet>
 <queryResultSet></queryResultSet>
</query>
```

The query string has been parsed and its components stored in three sections of the query structure. The queryWhereSet section encodes query constraint terms using post-fix notation. As can be seen in the example, the tokens TARGET_NAME, MARS, EQ have been included as queryElements along with tokens indicating their roles in the query. The elements included in this section, for example TARGET_NAME, will be used by the resource receiving the query to constrain the search.

The querySelectSet includes the names of elements to be return from the query. In this instance, since no fields have been specified, all available elements will be returned.

The queryFromSet is encoded similar to the queryWhereSet, except that its elements are used to restrict resources that might receive the query. For example, a resource identifier, its type, or associated discipline could be provided to limit the resources that will receive this query.

The queryResultSet is used by the resource receiving the query to package the query results.

The Java class XMLQuery has been developed to manage query structures. An instance can be constructed by either providing a keyword query string, such "TARGET_NAME = MARS", or by providing an existing query structure in XML format. Additional constructors will be implemented for other query formats. Methods are available for accessing any of the query elements and for putting and getting results from the queryResultSet.

The query service "crawls" through multiple nodes in the directed graph of resource systems automatically, locating additional servers that can fulfill a request for a particular item in any number of datasets. The query service uses "spider" objects to execute queries on each profile's XML description. The spider objects are part of the scatter-gather approach: each object can run in its own thread of execution, maximizing the concurrency of multiple nodes in the system. The system scatters the spiders across nodes and gathers their results as they become available.

Figure 4 shows a Unified Modeling Language (UML) [2] sequence diagram for a typical search. In the diagram, objects are shown across the top with their lifelines dropping down as time increases. Rectangles over the lifelines depict when an object is active. Solid arrows show method calls on an object, while dashed arrows show returns from those calls. A user's query triggers the action at the Query Server object through its "execute search" method. The Query Server asks its root Profile Server object for any matches to the query. In response, the Profile Server returns three possible other Profile Servers that could contain matches (in addition to any dataset matches it itself has). Concurrently, the Query Server executes the same query on the other Profile Servers. As each server returns more information, the Query Server may query yet more and more servers. Finally, after traversing the digraph in this way, the Query Server returns the search results in an XML document.

The Java programming language simplifies development of concurrent programming such as that used in the query component. Java includes built-in keywords and library classes for threading. However, Sun's marketing phrase for Java, "Write Once, Run Anywhere," is more hype than reality. In order to encourage implementations of Java on a wide variety of computer hardware and operating systems, Sun underspecified details of Java's threading behavior. As a result, behavior of threaded programs vary from implementation to implementation. Sometimes, programs hang (deadlock) even though there is no obvious deadlock in the code as implemented.

The query service experiences such hanging behavior. When running in a Windows-based Java environment, multiple concurrent queries work correctly and quickly. But on a Linux-based environment, multiple threads performing queries hang the query component. Because the scatter-gather approach to making multiple concurrent queries is far more efficient than serially querying remote nodes, we plan on incorporating a deeper investigation of the code and of various Java virtual machines for the Linux platform.

Since the directed graph of resource systems is not necessarily acyclic, the query component must take care not to re-query profile nodes it has already visited, or else it could get caught in an infinite loop. The query component trivially prevents this by tracking a set of profiles it has queried so far.

Once the query component's spiders have completed their tasks, the query service can immediately return the resource profiles to the user or the query service can re-broadcast the query to the resources identified so that they can perform the query. In the former case, the query service assembles the results into a single query structure and returns it to the user. This scenario was illustrated in Figure 1 where XML_QUERY(4) is returned to the client .The user can access the results document directly or they can be translated into HTML for presentation within a web browser. In the web browser, hyperlinks and additional searches are set up automatically by the translation process that enables the end user to immediately fetch products or visit sites that contain the sought datasets.

If the query is re-broadcasted to the identified resources, a similar scenario takes place. Each resource performs the query, gathers the results into a query structure, and returns the query structure to the query service. The query service then assembles the results into a single query structure and returns it to the user. This scenario is also illustrated in Figure 1 where XML_QUERY(4) returns data results.

Once the query component's spiders have completed their tasks, the query service assembles the results into an XML document. The user can access the results document directly or it can be translated into HTML for presentation within a web browser. In the web browser, hyperlinks and additional searches are set up automatically by the translation process that enables the end user to immediately fetch products or visit sites that contain the sought datasets.

One possible extension that we are considering for the query service is to make it available via the HTTP standard. This would allow HTML pages to send XML queries through the resource location service and render results directly into the HTML document as previously mentioned.

IV. Profile Service

Instruments and experiments generate science data products that are archived in data systems. Unfortunately, these data systems are heterogeneous and there are few common standards for querying. This makes locating data across these systems very difficult. Scientists and researchers are typically required to visit each data system independently and use local tools in order to locate the data. The profile service that is part of the framework uses metadata[1] to describe the data resources that exist within each data system within a distributed environment. It refocuses the problem of interoperability between data systems on metadata development and enables interoperability by using a common metadata interchange language.

The purpose of an OODT profile is to provide a resource description that is sufficient to determine if the resource can resolve a query. It is used by the OODT resource location service to identify and locate resources within the digraph and subsequently limit the number of resources that will have to consider the query. For example, within a space sciences implementation of this concept, a query for images of Jupiter should not have to be handled by the resource maintaining the Mars Global Surveyor images of Mars. A profile can be defined as a proper subset of the metadata that describes a resource and which is sufficient to determine whether the resource could resolve a query.

The Extensible Markup Language (XML) was chosen as the common interchange language for the OODT profiles. The advantages of XML include (1) superior expressiveness to HTML by allowing information-structure specifications, (2) simplicity compared to SGML in use and syntax, (3) wide acceptance as an Electronic Data Interchange (EDI) standard, and 4) the most compelling, its flexibility.

[1] Metadata is, literally, data about data, or information that describes the characteristics of data. For example, 37.6 is data. The fact that it's a measurement of a body's temperature in Kelvins is metadata. [6]

The flexibility of XML allowed us to develop a generic structure for managing resource descriptions from any science domain.

We defined the XML Extensible Profile Language (X2PL). X2PL provides both a means for capturing resource attributes as well as a language for capturing the attributes of the information content that the resource manages. The Document Type Definition (DTD) specification in Figure 5 illustrates the basic components of the resource profile. The DTD specification has three parts: the profile attributes, resource attributes, and the profile elements. We're using a DTD specification since the technology is readily understood and widely supported. We will consider XML-SCHEMA when it becomes a recommendation of the World Wide Web Consortium (W3C).

The profile itself is described in the profAttributes section using the attributes shown. The profId attribute provides a system-wide unique identifier for a profile instance. The profTitle and profDesc attributes provide descriptions of the profile where profTitle is more terse and appropriate for more frequent display in user interfaces. The profType attribute, which is further defined below, identifies the profile subtype. The profDataDictId attribute provides the identifier of the controlling domain data dictionary. The profChildId and profParentId attributes allow for the creation of a profile hierarchy.

```
<!ELEMENT profiles (profile+)>

<!ELEMENT profile
(profAttributes,
 resAttributes,
 profElement*)>

  <!ELEMENT profAttributes
  (profId, profVersion*, profTitle*,
    profDesc*, profType*, profStatusId*,
    profSecurityType*, profParentId*,
    profChildId*, profRegAuthority*,
    profRevisionNote*, profDataDictId*)>

  <!ELEMENT resAttributes
  (Identifier, Title*, Format*,
   Description*,Creator*, Subject*,
   Publisher*, Contributor*, Date*,
   Type*, Source*, Language*,
   Relation*, Coverage*, Rights*,
   resContext*, resClass*, resLocation*)>
```

```
<!ELEMENT profElement
(elemId, elemName*, elemDesc*,
 elemType*, elemUnit*,
 (elemValue |
  (elemMinValue, elemMaxValue))*,
 elemSynonym*, elemObligation*,
  elemMaxOccurrence*,elemComment*)>
```

Figure 5: OODT Profile DTD

The resource is described in the resAttributes section. Since the Dublin Core initiative has defined an internationally accepted metadata element set for describing electronic resources, its 15 recommended elements have been adopted. A description of this initiative and the elements is available at http://purl.org/DC/ . The OODT has added three additional resource attributes. The resContext element identifies the application environment or discipline within which the resource originates and is derived from a discipline taxonomy. For example, NASA.PDS.GEOSCIENCE is used to indicate the resource is associated with the Geoscience node of the Planetary Data System. The resLocation provides the location of the resource, typically represented as a URL. The resClass element identifies the resource within a resource taxonomy. Examples values are system.productServer, application.interface. data.granule, and data.dataSet.

The profile element section, the third part of the profile, describes the data content that the resource manages. For example, the Planetary Data System (PDS) maintains an inventory of all science data sets that have been archived in the system. Within this inventory, the data sets are indexed on the instrument that collected the data and the target body that the data was collected from. The profile element section would include these indexed attributes as data elements. Figure 6 shows a portion of the DIS resource profile. The attributes TARGET_NAME and DATA_OBJECT_TYPE are encoded into the profile element section, indicating that this resource, the PDS DIS, can handle queries on TARGET_NAME.

As can be seen from Figures 5 and 6, each data element is defined using the meta-attributes elemId and elemValue. To maintain compliance with developing international standards, these meta-attributes are consistent with those defined in ISO/IEC 11179 – Specification and Standardization of Data Elements. The description of this standard and the proposed elements is available at http://www.sdct.itl.nist.gov/~ftp/l8/other/coalition/Coalition.htm .

The basic profile structure as illustrated in Figure 5, actually has three subtypes. This is indicated by the value of profType. As is apparent, the profile element section is essentially a data dictionary. Because of this fact and the need to address requirements at three levels of aggregation, the profile structure was specialized into three subtypes: profile, dataDictionary, and inventory.

The profile subtype is used to describe one resource and for this specialization use of the meta-attributes elemDesc and elemSynonym are optional. A data dictionary would be the more appropriate location for these attributes.

```
<profile>
  <profAttributes>
    <profId>PROFILE_PDS_DIS</profId>
    <profTitle>DIS Profile</profTitle>
    <profDesc>This profile … </profDesc>
    <profType>profile</profType>
   <profDataDictId>PDS_DD
                       </profDataDictId>
  </profAttributes>
  <resAttributes>
    <Identifier>PDS DIS</Identifier>
    <Title>Distributed Inven …</Title>
    <Format>text/html</Format>
    <Language>en</Language>
    <resContext>NASA.PDS</resContext>
    <resClass>application.inventory
                        </resClass>
    <resLocation>http://…pdsbrows.htm
                      </resLocation>
  </resAttributes>
    <profElement>
      <elemId>DATA_OBJECT_TYPE</elemId>
      <elemType>ENUMERATION</elemType>
      <elemValue>CUBE</elemValue>
      <elemValue>IMAGE</elemValue>
      <elemValue>…</elemValue>
    </profElement>
    <profElement>
      <elemId>TARGET_NAME</elemId>
      <elemType>ENUMERATION</elemType>
      <elemValue>DEIMOS</elemValue>
      <elemValue>MARS</elemValue>
      <elemValue>PHOBOS</elemValue>
      <elemSynonym>ADS.OBJECT_ID
                      </elemSynonym>
    </profElement>
</profile>
```

Figure 6: Example Profile - PDS DIS

The next specialization of the profile structure is dataDictionary. For this specialization all meta-attributes including elemDesc and elemSynonym are appropriate and required. In addition, the preferred values of any enumerated type are the union of all preferred values over the domain. For example, in the data dictionary for the planetary science community, the TARGET_NAME data element would have a set of elemValue containing the names of all planets, satellites, comets, and asteroids.

The final specialization of the profile structure is the inventory. This profile subtype is a slight variation on profile and is used to describe individual data products. For example, an inventory could be used to profile all the Mars Global Surveyor images of Mars. This specialization would minimize space by eliminating many of the meta-attributes such as elemDesc.

The final phase of profile development involves implementing instances of profiles for specific domains. As is evident, the success of the distributed resource location concept is dependent on the existence of domain metadata captured in repositories such as data (element) dictionaries. Within such privileged domains, the registration of resources with the service is readily accomplished by extracting the necessary metadata from the domain's metadata repository, creating the resource profile, and then registering the profile with the service. This enables the successful location of resources within a domain.

The collection of profiles from several domains would enable the location of resources across domains, and even raises the possibility of resource interoperability.

Supporting interoperability between resources from different domains is strongly dependent on metadata compatibility, or how well the metadata spans the domains. For example, two related domains such as planetary science and astrophysics both associate one or more target bodies with most data products. However, unless the same identifiers are used for a specific target or a mapping between identifiers is determined, the attribute will not support resource location much less interoperability across the domains. In fact as more sophisticated interoperability such as data transformation and correlation are requested, deeper levels of metadata compatibility will be required. For example, once a target body is identified, sufficient metadata must be available for coordinate system conversion.

The profile, as a set of resource attributes, lends itself in an interesting way to the task of distributed resource location across heterogeneous domains. When considered from an object-oriented perspective, a resource has three modes. These are (1) the description of the resource as represented by the resource's attribute values, (2) the instance of the resource which is obtained by dereferencing the value of the resource's location attribute, and (3) the class definition as represented by the list of resource attributes and data elements in the profile element section.

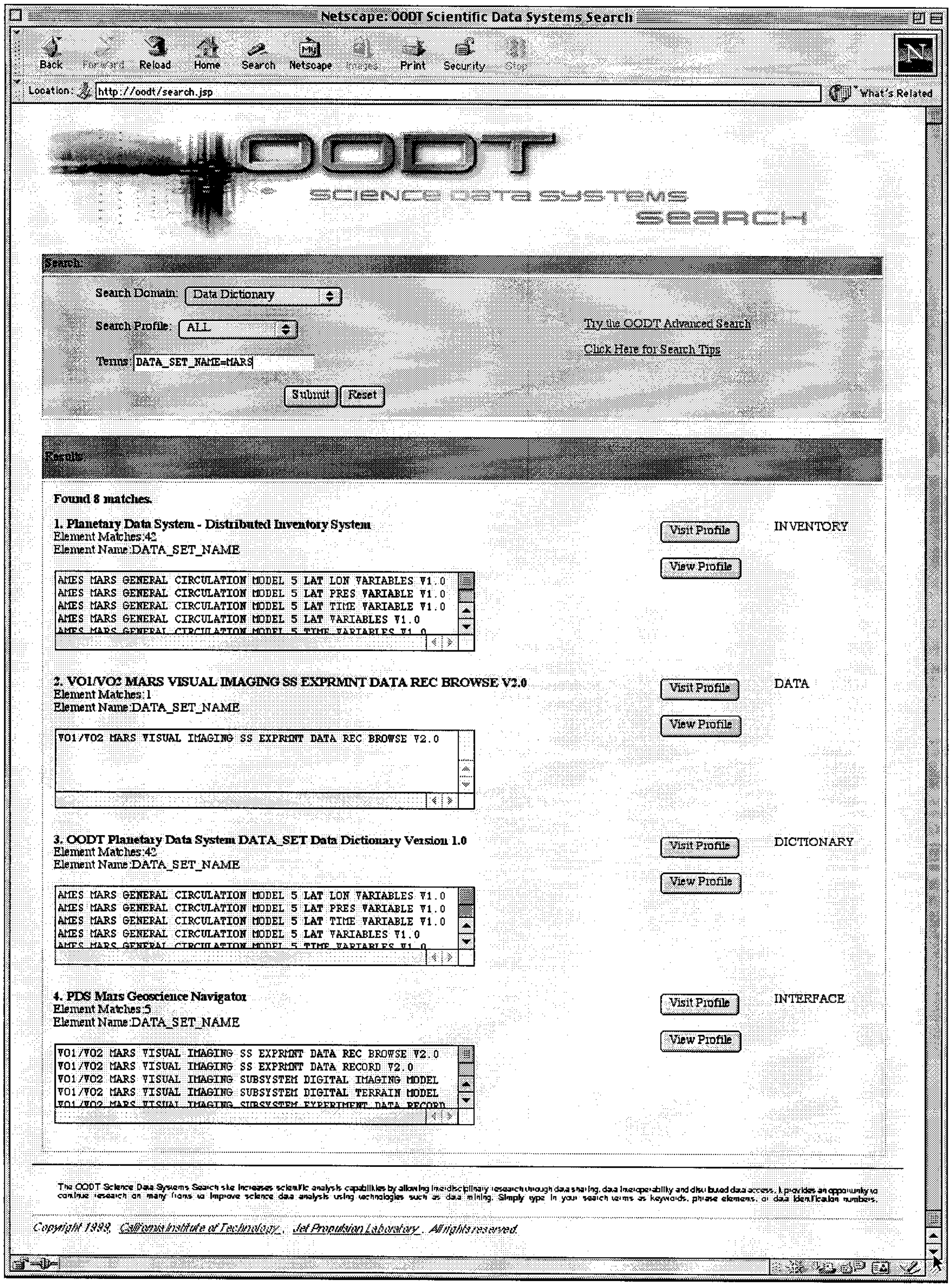
Within the resource location service, a query can be made for any of the three modes. For example, the primary role of a profile is to provide the location of a resource that can resolve a user query. Once identified, the value of resLocation is returned with any other selected profile attributes. As mentioned, the instance of a resource is obtained by de-referencing the value of the resLocation attribute. If a resource profile describes an HTML interface, the query could return the actual HTML page by performing a redirection on the value of resLocation. Finally, the class definition of a resource can be returned as a guide as to how the resources can be constrained in a query.

When using metadata to enable interoperability between domains, the hard problem of finding metadata commonalities across domains arises. This typically involves identifying similar attributes, determining core concepts, possibly generalizing the concept, and determining the key name and aliases. The resource location service has started to address this problem through the use of the data dictionary specialization and the use elemSynonym.

V. Product Service

The product service component, like the profile component, is instantiated as a node in the distributed architecture and provides the capability to return data system products based on a query. This allows each data system to maintain heterogeneous implementations, but still integrate into the enterprise architecture.

Each product server node provides the data access to one or more data systems. A product server node instantiates a Java-based server that integrates with the query service and receives XML-based queries using the XML query structure explained in Section III as part of the Query Service. The product server framework that is provided is a generic Java-based server that dynamically loads query handlers defined and registered with the service. Once a query is received by the framework it then notifies each registered query handler as a separate thread managed by the product server. This allows the product server to time out queries to resources which may not be available. The product server then packages the results from each query handler and returns the results using the XML-defined query definition. These results are then passed back to the Query Service which integrates all the results from the distributed product servers.



Location: http://oodt/search.jsp

Search Domain: Data Dictionary

Search Profile: ALL

Terms: DATA_SET_NAME=MARS

Try the OODT Advanced Search

Click Here for Search Tips

Found 8 matches.

1. Planetary Data System - Distributed Inventory System
Element Matches:42
Element Name:DATA_SET_NAME

INVENTORY

```
AMES MARS GENERAL CIRCULATION MODEL 5 LAT LON VARIABLES V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT PRES VARIABLE V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT TIME VARIABLE V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT VARIABLES V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 TIME VARIABLES V1.0
```

2. VO1/VO2 MARS VISUAL IMAGING SS EXPRMNT DATA REC BROWSE V2.0
Element Matches: 1
Element Name:DATA_SET_NAME

DATA

```
VO1/VO2 MARS VISUAL IMAGING SS EXPRMNT DATA REC BROWSE V2.0
```

3. OODT Planetary Data System DATA_SET Data Dictionary Version 1.0
Element Matches:42
Element Name:DATA_SET_NAME

DICTIONARY

```
AMES MARS GENERAL CIRCULATION MODEL 5 LAT LON VARIABLES V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT PRES VARIABLE V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT TIME VARIABLE V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 LAT VARIABLES V1.0
AMES MARS GENERAL CIRCULATION MODEL 5 TIME VARIABLES V1.0
```

4. PDS Mars Geoscience Navigator
Element Matches:5
Element Name:DATA_SET_NAME

INTERFACE

```
VO1/VO2 MARS VISUAL IMAGING SS EXPRMNT DATA REC BROWSE V2.0
VO1/VO2 MARS VISUAL IMAGING SS EXPRMNT DATA RECORD V2.0
VO1/VO2 MARS VISUAL IMAGING SUBSYSTEM DIGITAL IMAGING MODEL
VO1/VO2 MARS VISUAL IMAGING SUBSYSTEM DIGITAL TERRAIN MODEL
VO1/VO2 MARS VISUAL IMAGING SUBSYSTEM EXPERIMENT DATA RECORD
```

The OODT Science Data Systems Search site increases scientific analysis capabilities by allowing interdisciplinary research through data sharing, data interoperability and distributed data access. It provides an opportunity to continue research on many fronts to improve science data analysis using technologies such as data mining. Simply type in your search terms as keywords, phrase elements, or data identification numbers.

Copyright 1999, California Institute of Technology, Jet Propulsion Laboratory. All rights reserved.

Figure 8. Web interface showing a simple search using the resource location service.

Figure 8 demonstrates a query transaction which returns a list of products that are available from various product servers.

Query handlers provide a wrapper around each data system interface. This abstracts the data system away from the enterprise and allows the query handlers to function as a translation service. Developers implement query handlers using Java's type model, which separates types from classes using interfaces[2]. The query component specifies a standard Java interface to which query handlers must conform. Developers creating query handlers define classes that implement the query handler interface allowing the product framework to communicate with the query handlers.

The query handlers are loaded by the product server and passed the XML query. The query handlers transform the queries into the system-dependent query language in order to access the proprietary interface. This moves the responsibility for integrating the data-system–dependent data model onto the data system and away from the OODT data infrastructure. This is an important design consideration for accommodating scalability in a larger enterprise. An example would be the JPL central PDS node. Implementation of a query handler for this node requires that a mapping between the resource location service XML-based query and the central node's Sybase RDBMS be implemented. The query handler would then translate XML-based queries into a SQL-based query referencing the schema that was implemented by the PDS central node. This then provides the core mapping necessary to allow unique data system products to be retrieved from their native environments.

Once products are received by the query handler they must be transformed into a standard format that can be exchanged. The XML query structure defines the result format which allows for data to be returned in various formats. One of the requirements of this architecture is to provide a list of common interchange formats that imposes a set of standards for interoperability. The challenge is to provide a simple set of common formats for images and text, and require that results that fit into these categories use these formats for interchange. This would mean that all images that are in GIF may need to be converted into JPEG if that was the chosen format for images. It is important to point out that results which do not fit into a these standards can be returned in their native format.

[2] An interface in Java is a specification for the methods of a class. A class that implements a named interface must provide a definition for each method specified by interface or else be marked as an abstract class.

The goal is to provide flexibility in the architecture, but where possible promote standards for interoperability.

Product results are returned as ASCII text, or base-64 encoding depending on the data type. Base-64 encoding is used to return complex data products such as images. In many cases data systems may be able to return URL identifiers to data as results, rather than returning the complex data product. This improves performance by limiting the amount of data transported back to the client.

The product server design promotes interoperability by providing an interchange capability to allow a common query mechanism to retrieve products from unique data system implementations. The design presented allows distributed data system nodes to maintain their independence by providing a standard product server that can be extended to access the distributed data systems. This design provides a scalable solution by identifying a standard language for interoperability, and a framework for extending that interoperability to each data system. It also scales by pushing the implementation requirements onto each individual data system.

VI. Conclusion and Future Work

Products servers are presented as components of a distributed framework that allows heterogeneous data systems to communicate and share data. The components of this framework use XML, a standard metadata interchange language that has gained in popularity for improving the ability for applications to be integrated through electronic data interchange. This solution allows for loosely related data systems to remain distributed, while providing a content management and interchange capability for locating specific data products and resources archived at remote locations.

This component framework promotes the use of open standards. This architecture will accommodate changes as XML and standards for interoperability evolve. Currently, many organizations are looking at standards for electronic data interchange and queries using XML. At time of writing this paper W3C has just published a draft set of requirements for an XML query language [15].

Metadata really provides the foundation for our solution. The solution presented, although applied to planetary, astrophysics, and space science data problems, is not limited to those disciplines. In fact, the framework is adaptable based on the metadata definitions that are defined. This allows for the solution to then be applied to other disciplines including healthcare, defense, business, etc. Currently, we are also investigating use of this framework for locating and correlating physiologic and treatment data from pediatric research hospitals distributed across the United States. The benefit of the architecture is that it can easily accommodate different disciplines by refocusing the problem on metadata development. This means that industry and disciplines still must decide on metadata that includes common terms and definitions or at least mappings between terms that refer to the same things. Once this metadata is defined, the architecture provides means for allowing heterogeneous data systems to communicate so that advanced data discovery and mining techniques can be applied and new relationships discovered.

VII. References

1. Aho, A. V., Hopcroft, J. E., Ullman, J. D. Data Structures and Algorithms. Addison-Wesley 1983.

2. Booch et al. The Unified Modeling Language User Guide. Addison-Wesley. 1999.

3. Crichton, D.J., Hughes J.S., Hyon J.J., Kelly, S.C. Object Oriented Data Technology 1999 Annual Report. Interactive Analysis Environments Program. September 1999. http://oodt.jpl.nasa.gov/doc/reports/annual/1999

4. Data Entity Dictionary Specification Language (DEDSL) - Abstract Syntax, CCSDS 647.0-R-2.0, Draft Recommendation for Space Data System Standards, Consultative Committee on Space Data Systems, November 1999.

5. Devlin, B. Data Warehouse from Architecture to Implementation. Addison-Wesley. 1997.

6. Elmasri,R., Navathe,S., Fundamentals of Database Systems. The Benjamin/Cumings Publishing Company, Inc. 1994.

7. Hughes,J.S., Crichton,D.J., Hyon,J.J., Kelly,S.C., A Multi-Discipline Metadata Registry for Science Interoperability, Open Forum on Metadata Registries, ISO/IEC JTC1/SC32, Data Management and Interchange, January 2000, http://www.sdct.itl.nist.gov/~ftp/l8/sc32wg2/2000/events/openforum/index.htm

8. Hovy, E. Using Ontologies to Enable Access to Multiple Heterogeneous Databases, Open Forum on Metadata Registries, ISO/IEC JTC1/SC32, Data Management and Interchange, January 2000, http://www.sdct.itl.nist.gov/~ftp/l8/sc32wg2/2000/events/openforum/index.htm

9. Deutsch, A., Fernadez, M., Florescu, D., Levy, A., Suciu, D. XML-QL: A Query Language for XML. Submitted to W3C August 19, 1998. http://www.w3.org/TR/NOTE-xml-ql

10. Maruyama, H., Tamura, K., Uramoto, N. XML and Java: Developing Web Applications. Addison-Wesley. 1999.

11. Object Management Group. CORBA/IIOP 2.3.1 Specification. October 1999.

12. Orbacus for C++ and Java version 3.1.3. Object Oriented Concepts, Inc. 1999. http://ooc.com/

13. W3C. Document Object Model (DOM), Level 2 Specification. http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210.

14. W3C. Extensible Markup Language (XML), Version 1.0. http://www.w3.org/TR/1998/REC-xml-19980210.

15. W3C. XML Query Requirements. W3C Working Draft. 31 January 2000. http://www.w3.org/TR/2000/WD-xmlquery-req-20000131

16. Crichton, D.J., Hughes J.S., Hyon J.J., Kelly, S.C., Science Search and Retrieval using XML, The Second National Conference on Scientific and Technical Data, U.S. National Committee for CODATA, National Research Council, March 13-14, 2000.

http://oodt.jpl.nasa.gov/doc/papers/codata/paper.pdf.

VIII. About the Authors

Daniel Crichton is a Project Element Manager at JPL, and the principal investigator for the Object Oriented Data Technology task where he is leading a research task developing distributed frameworks for integrating science data management and archiving systems. He also currently serves as the implementation manager and architect of a JPL initiative to build an enterprise data architecture. His interests are in distributed architectures, enterprise and Internet technologies, and database systems. He holds a B.S. and M.S. in Computer Science. He can be reached at Daniel.J.Crichton@jpl.nasa.gov

Steven Hughes is a System Engineer at JPL, and a Co-Investigator for the Object Oriented Data Technology task. He is currently the technical lead engineer for the Planetary Data System and was instrumental in the development of the planetary science meta-model. His interests are in distributed architectures and the role of metadata in interoperability. He holds a B.S. and M.S. in Computer Science. He can be reached at Steven.Hughes@jpl.nasa.gov

Jason Hyon has been with JPL since 1985 and is currently a group supervisor for the Information Management Technology group. His research focuses on real time data architecture, information management, object-oriented distributed computing, and optical data storage. He is a principal investigator and a manager for the Integrated Information Management task for the JPL TMOD on-board data management research. He manages the Data Archival and Retrieval Enhancement (DARE) task for Defense Threat Reduction Agency, and NASA's Regional Planetary Imaging Facility. He is also a co-investigator for NASA's Data Distribution Lab and for the Object Oriented Data Technology task. He can be reached at Jason.Hyon@jpl.nasa.gov

Sean Kelly is a Senior Software Engineer at User Technology Associates and a consultant to JPL. He is currently supporting implementation for the Object Oriented Data Technology task. His interests include practical applications of software methods and leveraging web technologies in unique ways. He holds a B.S. in Computer Science and a B.S. in Technical Communication. He can be reached at Sean.Kelly@jpl.nasa.gov

IX. Copyright

The work described was performed at the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration.